고객을 세그먼테이션하자 [프로젝트]

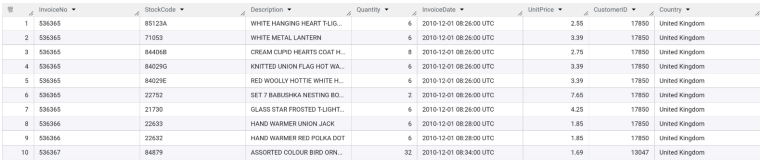
11-2. 데이터 불러오기

데이터 살펴보기

- **테이블에 있는 10개의 행만 출력하기**

```
SELECT *
FROM `proven-hash-466600-d5.modulabs_project.data`
LIMIT 10;
```

[결과]

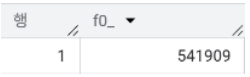


행	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1	536365	85123A	WHITE HANGING HEART T-LIG...	6	2010-12-01 08:26:00 UTC	2.55	17850	United Kingdom
2	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00 UTC	3.39	17850	United Kingdom
3	536365	844068	CREAM CUPID HEARTS COAT H...	8	2010-12-01 08:26:00 UTC	2.75	17850	United Kingdom
4	536365	84029G	KNITTED UNION FLAG HOT WA...	6	2010-12-01 08:26:00 UTC	3.39	17850	United Kingdom
5	536365	84029E	RED WOOLLY HOTTIE WHITE H...	6	2010-12-01 08:26:00 UTC	3.39	17850	United Kingdom
6	536365	22752	SET 7 BABUSHKA NESTING BO...	2	2010-12-01 08:26:00 UTC	7.65	17850	United Kingdom
7	536365	21730	GLASS STAR FROSTED T-LIGHT...	6	2010-12-01 08:26:00 UTC	4.25	17850	United Kingdom
8	536366	22633	HAND WARMER UNION JACK	6	2010-12-01 08:28:00 UTC	1.85	17850	United Kingdom
9	536366	22632	HAND WARMER RED POLKA DOT	6	2010-12-01 08:28:00 UTC	1.85	17850	United Kingdom
10	536367	84879	ASSORTED COLOUR BIRD ORN...	32	2010-12-01 08:34:00 UTC	1.69	13047	United Kingdom

- **전체 데이터는 몇 행으로 구성되어 있는지 확인하기**

```
SELECT COUNT(*)
FROM `proven-hash-466600-d5.modulabs_project.data`;
```

[결과]



행	f0_
1	541909

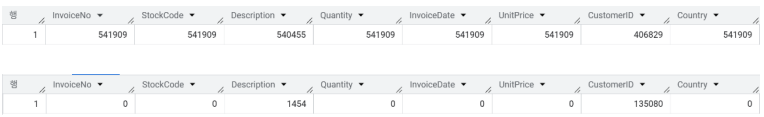
데이터 수 세기

- **COUNT 함수를 사용해서, 각 컬럼별 데이터 포인트의 수를 세어 보기**

```
-- 컬럼별 데이터 개수
SELECT COUNT(InvoiceNo) InvoiceNo,     COUNT(StockCode) StockCode,
      COUNT(Description) Description, COUNT(Quantity) Quantity,
      COUNT(InvoiceDate) InvoiceDate, COUNT(UnitPrice) UnitPrice,
      COUNT(CustomerID) CustomerID,   COUNT(Country) Country
FROM `proven-hash-466600-d5.modulabs_project.data`;

-- 컬럼별 NULL데이터 개수
SELECT COUNTIF(InvoiceNo IS NULL) InvoiceNo,     COUNTIF(StockCode IS NULL) StockCode,
      COUNTIF(Description IS NULL) Description, COUNTIF(Quantity IS NULL) Quantity,
      COUNTIF(InvoiceDate IS NULL) InvoiceDate, COUNTIF(UnitPrice IS NULL) UnitPrice,
      COUNTIF(CustomerID IS NULL) CustomerID,   COUNTIF(Country IS NULL) Country
FROM `proven-hash-466600-d5.modulabs_project.data`;
```

[결과]



행	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1	541909	541909	540455	541909	541909	541909	406829	541909

행	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1	0	0	1454	0	0	0	135080	0

11-4. 데이터 전처리 방법(1): 결측치 제거

컬럼 별 누락된 값의 비율 계산

- **각 컬럼 별 누락된 값의 비율을 계산**
  - **각 컬럼에 대해서 누락 값을 계산한 후, 계산된 누락 값을 UNION ALL을 통해 합치기**

```
SELECT 'InvoiceNo' AS InvoiceNo,
        ROUND(SUM(CASE WHEN InvoiceNo IS NULL THEN 1 ELSE 0 END) / COUNT(*) * 100, 2) AS missing_percentage
FROM `proven-hash-466600-d5.modulabs_project.data`
UNION ALL
SELECT 'StockCode' AS StockCode,
      ROUND(SUM(CASE WHEN StockCode IS NULL THEN 1 ELSE 0 END) / COUNT(*) * 100, 2) AS missing_percentage
FROM `proven-hash-466600-d5.modulabs_project.data`
UNION ALL
SELECT 'Description' AS Description,
      ROUND(SUM(CASE WHEN Description IS NULL THEN 1 ELSE 0 END) / COUNT(*) * 100, 2) AS missing_percentage
FROM `proven-hash-466600-d5.modulabs_project.data`
UNION ALL
SELECT 'Quantity' AS Quantity,
      ROUND(SUM(CASE WHEN Quantity IS NULL THEN 1 ELSE 0 END) / COUNT(*) * 100, 2) AS missing_percentage
FROM `proven-hash-466600-d5.modulabs_project.data`
UNION ALL
SELECT 'InvoiceDate' AS InvoiceDate,
      ROUND(SUM(CASE WHEN InvoiceDate IS NULL THEN 1 ELSE 0 END) / COUNT(*) * 100, 2) AS missing_percentage
FROM `proven-hash-466600-d5.modulabs_project.data`
UNION ALL
SELECT 'UnitPrice' AS UnitPrice,
      ROUND(SUM(CASE WHEN UnitPrice IS NULL THEN 1 ELSE 0 END) / COUNT(*) * 100, 2) AS missing_percentage
FROM `proven-hash-466600-d5.modulabs_project.data`
UNION ALL
SELECT 'CustomerID' AS CustomerID,
      ROUND(SUM(CASE WHEN CustomerID IS NULL THEN 1 ELSE 0 END) / COUNT(*) * 100, 2) AS missing_percentage
FROM `proven-hash-466600-d5.modulabs_project.data`
UNION ALL
SELECT 'Country' AS Country,
      ROUND(SUM(CASE WHEN Country IS NULL THEN 1 ELSE 0 END) / COUNT(*) * 100, 2) AS missing_percentage
FROM `proven-hash-466600-d5.modulabs_project.data`
```

[결과]

행	InvoiceNo	missing_percenta...
1	CustomerID	24.93
2	InvoiceDate	0.0
3	Quantity	0.0
4	InvoiceNo	0.0
5	Country	0.0
6	Description	0.27
7	UnitPrice	0.0
8	StockCode	0.0

결측치 처리 전략

- `StockCode = '85123A'` 의 `Description` 을 추출하는 쿼리문을 작성하기

```
SELECT DISTINCT Description
FROM `proven-hash-466600-d5.modulabs_project.data`
WHERE StockCode = '85123A'
ORDER BY Description;
```

[결과]

행	Description
1	?
2	CREAM HANGING HEART T-LIG...
3	WHITE HANGING HEART T-LIG...
4	wrongly marked carton 22804

결측치 처리

- **DELETE 구문을 사용하며, WHERE 절을 통해 데이터를 제거할 조건을 제시**

```
DELETE FROM `proven-hash-466600-d5.modulabs_project.data`
WHERE CustomerID IS NULL
  OR Description IS NULL;
```

[결과]

이 문으로 data의 행 135,080개가 삭제되었습니다.

```
SELECT COUNTIF(InvoiceNo IS NULL) InvoiceNo,     COUNTIF(StockCode IS NULL) StockCode,
      COUNTIF(Description IS NULL) Description, COUNTIF(Quantity IS NULL) Quantity,
      COUNTIF(InvoiceDate IS NULL) InvoiceDate, COUNTIF(UnitPrice IS NULL) UnitPrice,
      COUNTIF(CustomerID IS NULL) CustomerID,   COUNTIF(Country IS NULL) Country
FROM `proven-hash-466600-d5.modulabs_project.data`;
```

[결과]

행	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1	0	0	0	0	0	0	0	0

11-5. 데이터 전처리(2): 중복값 처리

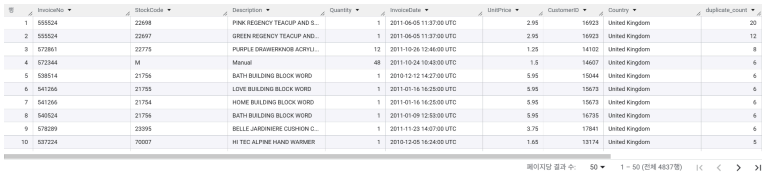
중복값 확인

- **중복된 행의 수를 세어보기**
  - **8개의 컬럼에 그룹 함수를 적용한 후, COUNT가 1보다 큰 데이터를 세어보기**

```
SELECT
  InvoiceNo, StockCode, Description, Quantity, InvoiceDate,
  UnitPrice, CustomerID, Country,
  COUNT(*) AS duplicate_count
FROM `proven-hash-466600-d5.modulabs_project.data`
GROUP BY
  InvoiceNo, StockCode, Description, Quantity, InvoiceDate,
  UnitPrice, CustomerID, Country
```

```
HAVING COUNT(*) > 1
ORDER BY duplicate_count DESC;
```

[결과]



행	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	duplicate_count
1	555524	22698	PINK REGENCY TEACUP AND S...	1	2011-06-05 11:37:00 UTC	2.95	16923	United Kingdom	20
2	555524	22697	GREEN REGENCY TEACUP AND...	1	2011-06-05 11:37:00 UTC	2.95	16923	United Kingdom	12
3	572861	22775	PURPLE DRAWERKNOB ACRYLI...	12	2011-10-26 12:46:00 UTC	1.25	14102	United Kingdom	8
4	572344	M	Manual	48	2011-10-24 10:43:00 UTC	1.5	14607	United Kingdom	6
5	538514	21756	BATH BUILDING BLOCK WORD	1	2010-12-12 14:27:00 UTC	5.95	15044	United Kingdom	6
6	541266	21755	LOVE BUILDING BLOCK WORD	1	2011-01-16 16:25:00 UTC	5.95	15673	United Kingdom	6
7	541266	21754	HOME BUILDING BLOCK WORD	1	2011-01-16 16:25:00 UTC	5.95	15673	United Kingdom	6
8	540524	21756	BATH BUILDING BLOCK WORD	1	2011-01-09 12:53:00 UTC	5.95	16735	United Kingdom	6
9	576289	23595	BELLE JARDINIERE CUSHION C...	1	2011-11-23 14:07:00 UTC	3.75	17841	United Kingdom	6
10	537224	70007	HI TEC ALPINE HAND WARMER	1	2010-12-05 16:24:00 UTC	1.65	13174	United Kingdom	5

페이지당 결과 수: 50 1 – 50 (전체 4837행)

중복값 처리

- **중복값을 제거하는 쿼리문 작성하기**
  - **CREATE OR REPLACE TABLE 구문을 활용하여 모든 컬럼(*)을 DISTINCT 한 데이터로 업데이트**

```
CREATE OR REPLACE TABLE `proven-hash-466600-d5.modulabs_project.data` AS
SELECT DISTINCT *
FROM `proven-hash-466600-d5.modulabs_project.data`;
```

[결과]

행	f0_
1	401604

11-6. 데이터 전처리(3): 오류값 처리

InvoiceNo 살펴보기

- **고유(unique)한 InvoiceNo의 개수를 출력하기**

```
SELECT COUNT(DISTINCT InvoiceNo)
FROM `proven-hash-466600-d5.modulabs_project.data`;
```

[결과]

행	unique_InvoiceNo_CNT
1	22190

- **고유한 InvoiceNo를 앞에서부터 100개를 출력하기**

```
SELECT DISTINCT InvoiceNo
FROM `proven-hash-466600-d5.modulabs_project.data`
LIMIT 100;
```

[결과]

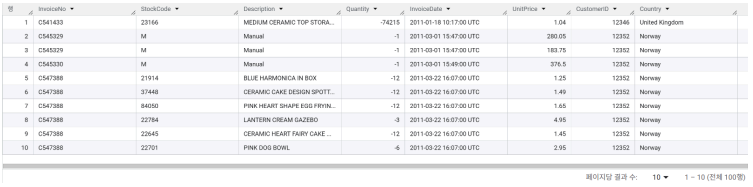
행	InvoiceNo
1	541431
2	C541433
3	537626
4	542237
5	549222
6	556201
7	562032
8	573511
9	581180
10	539318

페이지당 결과 수: 10 1 – 10 (전체 100행)

- `InvoiceNo` 가 'C'로 시작하는 행을 필터링 할 수 있는 쿼리문을 작성하기 (100행까지만 출력)

```
SELECT *
FROM `proven-hash-466600-d5.modulabs_project.data`
WHERE InvoiceNo LIKE 'C%'
LIMIT 100;
```

[결과]



- 구매 건 상태가 `Canceled` 인 데이터의 비율(%) - 소수점 첫번째 자리까지

```
-- COUNTIF
SELECT
  ROUND(COUNTIF(InvoiceNo LIKE 'C%') * 100.0 / COUNT(*), 1) AS canceled_ratio
FROM `proven-hash-466600-d5.modulabs_project.data`;

-- CASE WHEN-소
SELECT
  ROUND(SUM(CASE WHEN InvoiceNo LIKE 'C%' THEN 1 ELSE 0 END) * 100 / COUNT(*), 1) AS canceled_ratio
FROM `proven-hash-466600-d5.modulabs_project1.data`;
```

[결과]

행	canceled_ratio
1	2.2

`StockCode` 살펴보기

- 고유한 `StockCode` 의 개수를 출력하기

```
SELECT COUNT(DISTINCT StockCode) AS unique_stockcode_cnt
FROM `proven-hash-466600-d5.modulabs_project.data`;
```

[결과]

행	unique_stockcode_count
1	3684

- **어떤 제품이 가장 많이 판매되었는지 보기 위하여 `StockCode` 별 등장 빈도를 출력하기**
  - 상위 10개의 제품들을 출력하기

```
SELECT StockCode, COUNT(*) AS sell_cnt
FROM `proven-hash-466600-d5.modulabs_project.data`
GROUP BY StockCode
ORDER BY sell_cnt DESC
LIMIT 10;
```

[결과]

행	StockCode	sell_cnt
1	85123A	2065
2	22423	1894
3	85099B	1659
4	47566	1409
5	84879	1405
6	20725	1346
7	22720	1224
8	POST	1196
9	22197	1110
10	23203	1108

- `StockCode`**의 컬럼에 있던 값 중에서 숫자를 제외한 문자만 남기고 문자가 몇 자리 수 인지 세고**
  - **숫자가 0~1개인 값**들에는 어떤 코드들이 들어가 있는지 출력하기

```
WITH UniqueStockCodes AS (
  SELECT DISTINCT StockCode
  FROM `proven-hash-466600-d5.modulabs_project.data`)
SELECT
  LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS number_count,
  COUNT(*) AS stock_cnt
FROM UniqueStockCodes
GROUP BY number_count;

SELECT DISTINCT StockCode, number_count
FROM (SELECT StockCode,
                LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS number_count
        FROM project_name.modulabs_project.data)
WHERE number_count <= 1;
```

[결과]

행	number_count	stock_cnt
1	5	3676
2	0	7
3	1	1

행	StockCode	number_count
1	POST	0
2	M	0
3	C2	1
4	D	0
5	BANK CHARGES	0
6	PADS	0
7	DOT	0
8	CRUK	0

- `StockCode` **의 컬럼에 있던 값 중에서 숫자를 제외한 문자만 남기고 문자가 몇 자리 수 인지 세고**
  - **숫자가 0~1개인 값들을 가지고 있는 데이터 수는 전체 데이터 수 대비 몇 퍼센트**인지 구하기 (소수점 두 번째 자리까지)

```
WITH Annotated AS (
  SELECT *,
        LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS number_count
  FROM `proven-hash-466600-d5.modulabs_project.data`
)
SELECT
  ROUND(COUNTIF(number_count <= 1) * 100.0 / COUNT(*), 2) AS percent_missingdata
FROM Annotated;
```

[결과]

행	percent_missingd...
1	0.48

- **제품과 관련되지 않은 거래 기록을 제거하기**

```
DELETE FROM `proven-hash-466600-d5.modulabs_project.data`
WHERE StockCode IN ('POST', 'BANK CHARGES', 'ADJUST', 'C2', 'M', 'AMAZONFEE', 'DOT', 'CRUK');
```

[결과]

이 문으로 data의 행 1,834개가 삭제되었습니다.

Description 살펴보기

- **고유한 Description 별 출현 빈도를 계산하고 상위 30개를 출력하기**

```
SELECT Description, COUNT(*) AS description_cnt
FROM `proven-hash-466600-d5.modulabs_project.data`
GROUP BY Description
ORDER BY description_cnt DESC
LIMIT 30;
```

[결과]

행	Description	description_cnt
1	WHITE HANGING HEART T-LIG...	2058
2	REGENCY CAKESTAND 3 TIER	1894
3	JUMBO BAG RED RETROSPOT	1659
4	PARTY BUNTING	1409
5	ASSORTED COLOUR BIRD ORN...	1405
6	LUNCH BAG RED RETROSPOT	1345
7	SET OF 3 CAKE TINS PANTRY D...	1224
8	LUNCH BAG BLACK SKULL.	1099
9	PACK OF 72 RETROSPOT CAKE ...	1062
10	SPOTTY BUNTING	1026

페이지당 결과 수: 10 1 – 10 (전체 30행)

- **서비스 관련 정보를 포함하는 행들을 제거하기**

```
DELETE
FROM `proven-hash-466600-d5.modulabs_project.data`
WHERE REGEXP_CONTAINS(UPPER(Description), r'CARRIAGE|CHARGE|POSTAGE|ADJUST|COMMISSION|MANUAL|IMAGE|FEE
  OR REGEXP_CONTAINS(Description, r'\b(cm|g|ml|x\s?\d+|SIZE|DIMENSION|WEIGHT)\b');
```

[결과]

이 문으로 data의 행 4,174개가 삭제되었습니다.

- **대소문자를 혼합하고 있는 데이터를 대문자로 표준화 하기**

```
CREATE OR REPLACE TABLE `proven-hash-466600-d5.modulabs_project.data` AS
SELECT
  * EXCEPT (Description),
  UPPER(Description) AS Description
FROM `proven-hash-466600-d5.modulabs_project.data`;
```

[결과]

이 문으로 이름이 data인 테이블이 교체되었습니다.

UnitPrice 살펴보기

- **UnitPrice의 최솟값, 최댓값, 평균을 구하기**

```
SELECT
  MIN(UnitPrice) AS min_price,
  MAX(UnitPrice) AS max_price,
  ROUND(AVG(UnitPrice), 2) AS avg_price
FROM `proven-hash-466600-d5.modulabs_project.data`;
```

[결과]

행	min_price	max_price	avg_price
1	0.0	1867.86	2.92

- **단가가 0원인 거래의 개수, 구매 수량(Quantity)의 최솟값, 최댓값, 평균 구하기**

```
SELECT
  COUNT(*) AS cnt_quantity,
  MIN(Quantity) AS min_quantity,
  MAX(Quantity) AS max_quantity,
  ROUND(AVG(Quantity), 2) AS avg_quantity
FROM `proven-hash-466600-d5.modulabs_project.data`
WHERE UnitPrice = 0;
```

[결과]

행	cnt_quantity	min_quantity	max_quantity	avg_quantity
1	33	1	12540	420.06

- `UnitPrice = 0` **를 제거하고 일관된 데이터셋을 유지하기**

```
CREATE OR REPLACE TABLE project_name.modulabs_project.data AS
SELECT *
FROM `proven-hash-466600-d5.modulabs_project.data`
WHERE UnitPrice > 0;
```

[결과]

이 문으로 이름이 data인 테이블이 교체되었습니다.

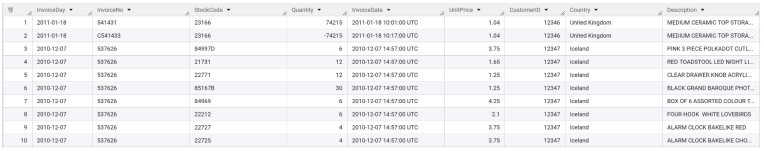
11-7. RFM 스코어

Recency

- `InvoiceDate` **컬럼을 연월일 자료형으로 변경하기**

```
SELECT DATE(InvoiceDate) AS InvoiceDay, *
FROM `proven-hash-466600-d5.modulabs_project.data`;
```

[결과]



- **가장 최근 구매 일자를 MAX() 함수로 찾아보기**

```
SELECT  DATE(InvoiceDate) AS InvoiceDay,
            most_recent_date
FROM `proven-hash-466600-d5.modulabs_project.data`
CROSS JOIN (SELECT MAX(DATE(InvoiceDate)) AS most_recent_date
                FROM `proven-hash-466600-d5.modulabs_project.data`);
```

[결과]

행	InvoiceDay	most_recent_date
1	2011-01-18	2011-12-09
2	2011-01-18	2011-12-09
3	2010-12-07	2011-12-09
4	2010-12-07	2011-12-09
5	2010-12-07	2011-12-09
6	2010-12-07	2011-12-09
7	2010-12-07	2011-12-09
8	2010-12-07	2011-12-09
9	2010-12-07	2011-12-09
10	2010-12-07	2011-12-09

- **유저 별로 가장 큰 InvoiceDay를 찾아서 가장 최근 구매일로 저장하기**

```
SELECT CustomerID,
       MAX(DATE(InvoiceDate)) AS InvoiceDay
FROM `proven-hash-466600-d5.modulabs_project.data`
GROUP BY CustomerID;
```

[결과]

행	CustomerID	InvoiceDay
1	12346	2011-01-18
2	12347	2011-12-07
3	12348	2011-09-25
4	12349	2011-11-21
5	12350	2011-02-02
6	12352	2011-11-03
7	12353	2011-05-19
8	12354	2011-04-21
9	12355	2011-05-09
10	12356	2011-11-17

- **가장 최근 일자(`most_recent_date`)와 유저별 마지막 구매일(`InvoiceDay`)간의 차이를 계산하기**

```
SELECT CustomerID,
        EXTRACT(DAY FROM MAX(InvoiceDay) OVER () - InvoiceDay) AS recency
FROM (SELECT CustomerID,
             MAX(DATE(InvoiceDate)) AS InvoiceDay
      FROM `proven-hash-466600-d5.modulabs_project`.data
      GROUP BY CustomerID);
```

[결과]

행	CustomerID	recency
1	12349	18
2	12379	81
3	12412	74
4	12490	5
5	12564	261
6	12607	58
7	13017	7
8	13044	291
9	13229	225
10	13255	3

- **최종 데이터 셋에 필요한 데이터들을 각각 정제해서 이어붙이고 지금까지의 결과를 `user_r` 이라는 이름의 테이블로 저장하기**

```
CREATE OR REPLACE TABLE `proven-hash-466600-d5.modulabs_project.user_r` AS
SELECT CustomerID,
        EXTRACT(DAY FROM MAX(InvoiceDay) OVER () - InvoiceDay) AS recency
FROM (SELECT CustomerID,
             MAX(DATE(InvoiceDate)) AS InvoiceDay
      FROM `proven-hash-466600-d5.modulabs_project.data`
      GROUP BY CustomerID);
```

[결과]

행	CustomerID	recency
1	17001	0
2	15910	0
3	14422	0
4	13069	0
5	18102	0
6	17754	0
7	17428	0
8	14446	0
9	13113	0
10	12985	0

Frequency

- **고객마다 고유한 InvoiceNo의 수를 세어보기**

```
SELECT CustomerID,
        COUNT(DISTINCT InvoiceNo) AS purchase_cnt
FROM `proven-hash-466600-d5.modulabs_project.data`
GROUP BY CustomerID;
```

[결과]

행	CustomerID	purchase_cnt
1	12346	2
2	12347	7
3	12348	4
4	12349	1
5	12350	1
6	12352	8
7	12353	1
8	12354	1
9	12355	1
10	12356	3

- **각 고객 별로 구매한 아이템의 총 수량 더하기**

```
SELECT CustomerID,
        SUM(Quantity) AS item_cnt
FROM `proven-hash-466600-d5.modulabs_project.data`
GROUP BY CustomerID;
```

[결과]

행	CustomerID	item_cnt
1	12346	0
2	12347	2458
3	12348	2332
4	12349	630
5	12350	196
6	12352	459
7	12353	20
8	12354	530
9	12355	240
10	12356	1541

- **전체 거래 건수 계산와 구매한 아이템의 총 수량 계산의 결과를 합쳐서 `user_rf` 라는 이름의 테이블에 저장하기**

```
CREATE OR REPLACE TABLE `proven-hash-466600-d5.modulabs_project.user_rf` AS

-- (1) 전체 거래 건수 계산
WITH purchase_cnt AS (
  SELECT
    CustomerID,
    COUNT(DISTINCT InvoiceNo) AS purchase_cnt
  FROM `proven-hash-466600-d5.modulabs_project.data`
  GROUP BY CustomerID
),

-- (2) 구매한 아이템 총 수량 계산
item_cnt AS (
  SELECT
    CustomerID,
    SUM(Quantity) AS item_cnt
  FROM `proven-hash-466600-d5.modulabs_project.data`
  GROUP BY CustomerID
)

-- 기존의 user_r에 (1)과 (2)를 통합
```

```
SELECT
  pc.CustomerID,
  pc.purchase_cnt,
  ic.item_cnt,
  ur.recency
FROM purchase_cnt AS pc
JOIN item_cnt AS ic
  ON pc.CustomerID = ic.CustomerID
JOIN `proven-hash-466600-d5.modulabs_project.user_r` AS ur
  ON pc.CustomerID = ur.CustomerID;
```

[결과]

행	CustomerID	purchase_cnt	item_cnt	recency
1	12713	1	505	0
2	13298	1	96	1
3	13436	1	76	1
4	17436	1	12	1
5	15520	1	314	1
6	14569	1	79	1
7	15195	1	1404	2
8	14204	1	72	2
9	15471	1	249	2
10	14578	1	240	3

Monetary

- **고객별 총 지출액 계산 (소수점 첫째 자리에서 반올림)**

```
SELECT
  CustomerID,
  ROUND(SUM(Quantity * UnitPrice), 1) AS user_total
FROM `proven-hash-466600-d5.modulabs_project1.data`
WHERE CustomerID IS NOT NULL
GROUP BY CustomerID;
```

[결과]

행	CustomerID	user_total
1	12346	0.0
2	12347	4310.0
3	12348	1437.2
4	12349	1457.5
5	12350	294.4
6	12352	1245.6
7	12353	89.0
8	12354	1079.4
9	12355	459.4
10	12356	2466.6

- **고객별 평균 거래 금액 계산**
  - **고객별 평균 거래 금액을 구하기 위해 1) `data` 테이블을 `user_rf` 테이블과 조인(LEFT JOIN) 한 후, 2) `purchase_cnt`로 나누어서 3) `user_rfm` 테이블로 저장하기**

```
CREATE OR REPLACE TABLE `proven-hash-466600-d5.modulabs_project.user_rfm` AS
SELECT
  rf.CustomerID AS CustomerID,
  rf.purchase_cnt,
  rf.item_cnt,
  rf.recency,
  ut.user_total,
  ROUND(ut.user_total / rf.purchase_cnt, 1) AS user_average
FROM `proven-hash-466600-d5.modulabs_project.user_rf` rf
LEFT JOIN (
  -- 고객별 총 지출액 계산 (Monetary)
  SELECT
    CustomerID,
    ROUND(SUM(Quantity * UnitPrice), 1) AS user_total
  FROM `proven-hash-466600-d5.modulabs_project.data`
  WHERE CustomerID IS NOT NULL
  GROUP BY CustomerID
) ut
ON rf.CustomerID = ut.CustomerID;
```

[결과]

행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average
1	12713	1	505	0	794.5	794.5
2	15520	1	314	1	343.5	343.5
3	13436	1	76	1	196.9	196.9
4	13298	1	96	1	360.0	360.0
5	14569	1	79	1	227.4	227.4
6	17436	1	12	1	17.4	17.4
7	14204	1	72	2	150.6	150.6
8	15195	1	1404	2	3861.0	3861.0
9	15471	1	249	2	442.1	442.1
10	12442	1	181	3	144.1	144.1

RFM 통합 테이블 출력하기

- **최종 user_rfm 테이블을 출력하기**

```
SELECT *
FROM `proven-hash-466600-d5.modulabs_project1.user_rfm`
```

[결과]

행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average
1	12713	1	505	0	794.5	794.5
2	15520	1	314	1	343.5	343.5
3	13436	1	76	1	196.9	196.9
4	13298	1	96	1	360.0	360.0
5	14569	1	79	1	227.4	227.4
6	17436	1	12	1	17.4	17.4
7	14204	1	72	2	150.6	150.6
8	15195	1	1404	2	3861.0	3861.0
9	15471	1	249	2	442.1	442.1
10	12442	1	181	3	144.1	144.1

11-8. 추가 Feature 추출

1. 구매하는 제품의 다양성

- **1) 고객 별로 구매한 상품들의 고유한 수를 계산하기**
  2) `user_rfm` **테이블과 결과를 합치기**
  3) `user_data` **라는 이름의 테이블에 저장하기**

```
CREATE OR REPLACE TABLE `proven-hash-466600-d5.modulabs_project.user_data` AS
WITH unique_products AS (
  SELECT
    CustomerID,
    COUNT(DISTINCT StockCode) AS unique_products
  FROM `proven-hash-466600-d5.modulabs_project.data`
  WHERE CustomerID IS NOT NULL
  GROUP BY CustomerID
)
SELECT
  ur.*,
  up.unique_products
FROM `proven-hash-466600-d5.modulabs_project.user_rfm` AS ur
JOIN unique_products AS up
ON ur.CustomerID = up.CustomerID;
```

[결과]

행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average	unique_products
1	18141	1	-12	360	-35.4	-35.4	1
2	16138	1	-1	368	-8.0	-8.0	1
3	16737	1	288	53	417.6	417.6	1
4	17986	1	10	56	20.8	20.8	1
5	16454	1	2	64	5.9	5.9	1
6	13829	1	-12	359	-102.0	-102.0	1
7	16765	1	4	294	34.0	34.0	1
8	15070	1	36	372	106.2	106.2	1
9	13017	1	48	7	204.0	204.0	1
10	15423	1	100	38	36.0	36.0	1

2. 평균 구매 주기

- **고객들의 쇼핑 패턴을 이해하는 것을 목표 (고객 별 재방문 주기 살펴보기)**
  - **균 구매 소요 일수를 계산하고, 그 결과를** `user_data` **에 통합**

```
CREATE OR REPLACE TABLE `proven-hash-466600-d5.modulabs_project1.user_data` AS
WITH purchase_intervals AS (
  -- (2) 고객 별 구매와 구매 사이의 평균 소요 일수
  SELECT
    CustomerID,
    IFNULL(ROUND(AVG(interval_), 2), 0) AS average_interval
  -- (1) 구매와 구매 사이에 소요된 일수
  FROM (
    SELECT
      CustomerID,
      DATE_DIFF(DATE(InvoiceDate), LAG(DATE(InvoiceDate)) OVER (PARTITION BY CustomerID ORDER BY DATE(InvoiceDate)), DAY
    FROM
      `proven-hash-466600-d5.modulabs_project1.data`
    WHERE CustomerID IS NOT NULL
  )
  GROUP BY CustomerID
)

SELECT
  u.* EXCEPT(average_interval),
```

```
  IFNULL(pi.average_interval, 0) AS average_interval
FROM `proven-hash-466600-d5.modulabs_project1.user_data` AS u
LEFT JOIN purchase_intervals AS pi
ON u.CustomerID = pi.CustomerID;
```

[결과]

행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average	unique_products	average_interval
1	14119	1	-2	354	-19.9	-19.9	1	0.0
2	15070	1	36	372	106.2	106.2	1	0.0
3	13703	1	10	318	99.5	99.5	1	0.0
4	17436	1	12	1	17.4	17.4	1	0.0
5	16257	1	1	176	21.9	21.9	1	0.0
6	12943	1	-1	301	-3.8	-3.8	1	0.0
7	18141	1	-12	360	-35.4	-35.4	1	0.0
8	13099	1	288	99	207.4	207.4	1	0.0
9	17763	1	12	263	15.0	15.0	1	0.0
10	13302	1	5	155	63.8	63.8	1	0.0

3. 구매 취소 경향성

- **고객의 취소 패턴 파악하기**
  1) 취소 빈도(cancel_frequency) : 고객 별로 취소한 거래의 총 횟수
  2) 취소 비율(cancel_rate) : 각 고객이 한 모든 거래 중에서 취소를 한 거래의 비율
  - **취소 빈도와 취소 비율을 계산하고 그 결과를 `user_data` 에 통합하기**
    (취소 비율은 소수점 두번째 자리)

```
CREATE OR REPLACE TABLE `proven-hash-466600-d5.modulabs_project1.user_data` AS

WITH TransactionInfo AS (
  SELECT
    CustomerID,
    COUNT(DISTINCT InvoiceNo) AS total_transactions,
    COUNT(DISTINCT IF(STARTS_WITH(InvoiceNo, 'C'), InvoiceNo, NULL)) AS cancel_frequency
  FROM `proven-hash-466600-d5.modulabs_project1.data`
  WHERE CustomerID IS NOT NULL
  GROUP BY CustomerID
)

SELECT
  u.*,
  t.* EXCEPT(CustomerID),
  ROUND(IFNULL(t.cancel_frequency / t.total_transactions, 0), 2) AS cancel_rate
FROM `proven-hash-466600-d5.modulabs_project1.user_data` AS u
LEFT JOIN TransactionInfo AS t
ON u.CustomerID = t.CustomerID;
```

[결과]

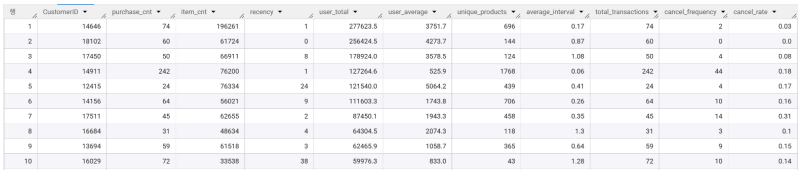
행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average	unique_products	average_interval	total_transactions	cancel_frequency	cancel_rate
1	16144	1	16	246	175.2	175.2	1	0.0	1	0	0.0
2	14705	1	100	198	179.0	179.0	1	0.0	1	0	0.0
3	16093	1	20	106	17.0	17.0	1	0.0	1	0	0.0
4	12659	1	104	29	73.7	73.7	4	0.0	1	0	0.0
5	15347	1	130	366	201.6	201.6	5	0.0	1	0	0.0
6	16106	1	74	65	108.4	108.4	6	0.0	1	0	0.0
7	15397	1	20	289	94.0	94.0	6	0.0	1	0	0.0
8	17879	1	45	173	178.5	178.5	6	0.0	1	0	0.0
9	17709	1	17	169	137.4	137.4	7	0.0	1	0	0.0
10	18086	1	106	274	101.3	101.3	7	0.0	1	0	0.0

- **다양한 컬럼들을 활용하여 고객의 구매 패턴과 선호도를 보다 심층적으로 이해할 수 있도록 최종적으로 `user_data` 를 출력하기**

```
SELECT *
FROM `proven-hash-466600-d5.modulabs_project1.user_data`
```

```
ORDER BY user_total DESC;
```

[결과]



행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average	unique_products	average_interval	total_transactions	cancel_frequency	cancel_rate
1	14646	74	196261	1	277623.5	3751.7	696	0.17	74	2	0.03
2	18102	60	61724	0	256424.5	4273.7	144	0.87	60	0	0.0
3	17450	50	66911	8	178924.0	3578.5	124	1.08	50	4	0.08
4	14911	242	76200	1	127264.6	525.9	1768	0.06	242	44	0.18
5	12415	24	76334	24	121540.0	5064.2	439	0.41	24	4	0.17
6	14156	64	56021	9	111603.3	1743.8	706	0.26	64	10	0.16
7	17511	45	62655	2	87450.1	1943.3	458	0.35	45	14	0.31
8	16684	31	48634	4	64304.5	2074.3	118	1.3	31	3	0.1
9	13694	59	61518	3	62465.9	1058.7	365	0.64	59	9	0.15
10	16029	72	33538	38	59976.3	833.0	43	1.28	72	10	0.14

회고

Keep : BigQuery를 활용해 대용량 데이터를 효율적으로 가공하고, 단계별 쿼리를 수행해 최종적으로 하나의 user_data 테이블에 통합해 분석용 데이터셋을 완성했다.

Problem : 데이터 삭제과정에서 실수가 있어 다시 데이터를 불러와 진행, 컬럼 중복 문제(average_interval) 에러가 발생할때마다 불필요하게 시간이 소요되어 아직 부족한점을 많이 느꼈다. 그래서 쿼리가 정상 실행되어도 계속 결과를 확인하고 의심하는 시간이 많았다.

Try : Recency, Frequency, Monetary + 부가 지표까지 한 번에 생성하는 SQL 쿼리도 시도하면 좋을 것 같고 시각화, 머신러닝 적용 등 내일 진행할 내용까지 포함하지 않아 아직 마무리가 덜 된 느낌... 무엇보다 아직 쿼리 작성에 익숙하지 않다는 생각이 계속 들어서 연습이 더 필요할 것 같다.